Flexible Execution of Partial Order Plans With Temporal Constraints*

Christian Muise[1], J. Christopher Beck[2], and **Sheila A. McIlraith[1]**

[1]Dept. of Computer Science
University of Toronto
{cjmuise,sheila}@cs.toronto.edu

[2]Dept. of Mechanical & Industrial Engineering
University of Toronto
jcb@mie.utoronto.ca

Abstract

We propose a unified approach to plan execution and schedule dispatching that converts a plan, which has been augmented with temporal constraints, into a policy for dispatching. Our approach generalizes the original plan and temporal constraints so that the executor need only consider the subset of state that is relevant to successful execution of valid plan fragments. We can accommodate a variety of calamitous and serendipitous changes to the state of the world by supporting the seamless re-execution or omission of plan fragments, without the need for costly replanning. Our methodology for plan generalization and online dispatching is a novel combination of plan execution and schedule dispatching techniques. We demonstrate the effectiveness of our method through a prototype implementation and a series of experiments.

1 Introduction

Plans and schedules often go awry because of unanticipated changes in the world. In such cases, it is up to the execution monitoring system (EM) to determine what to do. Typically, an EM represents the temporal plan it is executing as a partial-order plan (POP) with an associated simple temporal network (STN) [Dechter *et al.*, 1991] that captures the temporal constraints between actions [Younes and Simmons, 2003; Coles *et al.*, 2010]. The EM executes the POP's actions one after another until the goal is reached or a discrepancy is detected. Often, the EM is forced to resolve discrepancies through costly replanning, rescheduling, or plan repair (e.g., the IxTeT-eXeC system [Lemai and Ingrand, 2003]).

The focus of this paper is on maximizing the robustness of plan execution by minimizing the need for replanning. We propose an execution module, TPOPEXEC, which is comprised of two components: 1) COMPILER, an offline preprocessor that takes as input a POP and a set of temporal constraints, and produces a generalized representation; and 2) EXECUTOR, an online component that soundly selects a temporally consistent, *valid* plan fragment from the generalized plan. TPOPEXEC does no replanning or repair. Rather, it can serve as a component of a larger execution engine to reduce, but not eliminate, the need for replanning.

TPOPEXEC reacts to the state of the world, proposing the next action of one of a large number of valid plan fragments whose starting state satisfies the necessary conditions for plan validity. This enables TPOPEXEC to seamlessly elect to execute parts of a plan multiple times and/or to omit actions that are no longer necessary for achieving the goal.

Such flexibility can introduce ambiguity in the interpretation of temporal constraints. For example, if you must start eating 3 to 10 minutes after heating your dinner, and eating gets delayed causing you to re-heat, then what temporal relationship(s), if any, should exist between the first heating and the eating? As such ambiguities are not addressed by STNs, we introduce a specification language for temporal constraints that avoids the execution-time ambiguities and further supports the specification of constraints between state conditions and actions. We formally define the semantics of the temporal constraints and prove the correctness of TPOPEXEC. Compared to conventional EM systems, our approach has the potential to avoid replanning exponentially more often (in the size of the state). Experiments with a simulated uncertain environment show TPOPEXEC achieving the goal in 92% of the trials while the standard STN dispatching technique only has a success rate of roughly 30%.

TPOPEXEC leverages existing work from both partial-order plan execution and temporal reasoning. While many of the core algorithms are based on existing techniques, our main contribution stems from the dynamic creation of temporal subproblems that need to be solved during execution. Our approach is noteworthy for its novelty and broad applicability while making an important step towards integrating plan execution and schedule dispatching – tasks that are traditionally addressed independently [Smith *et al.*, 2000].

2 Preliminaries

STRIPS Following [Ghallab *et al.*, 2004], a *STRIPS Planning Problem* is a tuple $\Pi = \langle F, O, I, G \rangle$ where F is a set of fluent symbols, O is a set of action operators, and I and G are sets of fluents, corresponding to the initial state and goal condition. Every action $a \in O$ is defined by three sets of fluents PRE, ADD, and DEL, corresponding to the preconditions,

*This paper also appears in the Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013).

add effects, and delete effects. An action a is *executable* in state s iff $PRE(a) \subseteq s$. An executable sequence of actions is a *plan*. A plan that commences with I and terminates in G is a *valid plan* for G. Given a plan $a_0, \ldots, a_n$, the sequence of actions $a_i, \ldots, a_n$, where $i \geq 0$, is a *plan suffix*.

Partial-order Plans A partial-order plan (POP) is a tuple $P = \langle \mathcal{A}, \mathcal{O} \rangle$ where $\mathcal{A}$ is the set of actions in the plan and $\mathcal{O}$ is a set of orderings between the actions in $\mathcal{A}$ (e.g., $(a_1 \prec a_2) \in \mathcal{O}$) [Weld, 1994]. For this work, we do not require a set of causal links to be defined. A total ordering of the actions in $\mathcal{A}$ that respects $\mathcal{O}$ is a *linearization* of P. A POP provides a compact representation for multiple linearizations, and is considered *valid* iff every linearization is a valid plan. We further assume that the set of ordering constraints $\mathcal{O}$ in a valid POP is transitively closed. We typically add two special actions to the POP, a_I and a_G, that encode the initial and goal states through their add effects and preconditions. A *POP suffix* of a given POP $\langle \mathcal{A}, \mathcal{O} \rangle$ is any POP $\langle \mathcal{A}', \mathcal{O}' \rangle$ where (1) $\mathcal{A}' \subseteq \mathcal{A}$, (2) $\mathcal{O}' = \{(a_1 \prec a_2) \mid (a_1 \prec a_2) \in \mathcal{O}$ and $a_1, a_2 \in \mathcal{A}'\}$, and (3) $\forall a_1 \in \mathcal{A}', ((a_1 \prec a_2) \in \mathcal{O}) \rightarrow (a_2 \in \mathcal{A}')$. That is, every ordering originating from an action in the suffix implies the corresponding action is also in the suffix. While the same ground action may appear more than once in $\mathcal{A}$, we assume that every element of $\mathcal{A}$ is uniquely identifiable.

Durative Actions Following Fox and Long (2003), any durative action appears in the plan as a pair of instantaneous *start* and *end* actions that must alternate (a durative action must end before it can be started again). We further augment the pair of actions with a suitable temporal constraint to enforce the duration (cf. Section 3.1). We handle domains where durative actions must overlap, typically referred to as *required concurrency* [Fox *et al.*, 2004; Cushing *et al.*, 2007], but we do not handle situations where a single durative action must overlap with itself during execution [Coles *et al.*, 2008]: when a *start* action occurs, the corresponding *end* action must occur before the *start* can be executed again. Our work is focused on STRIPS planning problems and a set of temporal constraints inspired by those found in PDDL3.0. STRIPS cannot require that two instantaneous actions "execute in parallel" (e.g., [Boutilier and Brafman, 2001]), so this situation will not arise. In the future, we hope to expand to other aspects of PDDL including conditional effects, numeric state fluents, and continuous change.

3 Temporal Constraints and Traces

The starting point for our work is a *Temporally Constrained POP* (TPOP), $\langle \langle \mathcal{A}, \mathcal{O} \rangle, \mathcal{C} \rangle$, which comprises a valid POP $\langle \mathcal{A}, \mathcal{O} \rangle$, and a set of temporal constraints $\mathcal{C}$. Here, we do not concern ourselves with where the POP comes from, but options include using a partial-order planner (e.g., VHPOP [Younes and Simmons, 2003]), relaxing a sequential plan (e.g., [Muise *et al.*, 2012]), or having a POP specified by the user. Temporal constraints originate with the user, with the exception of those generated in the transformation of durative actions to pairs of instantaneous actions. This decoupling enables a POP to be re-used in multiple scenarios by simply varying the temporal constraints. In this section, we propose a syntax and semantics for the temporal constraints in $\mathcal{C}$.

Consider the TPOPEXEC for a mobile-phone based cognitive assistant (CA) that oversees a user's daily activities. The CA knows his/her plan for the day including activities such as laundry, transportation, dinner, and a movie. It is updated about the state of the world by the user, RSS feeds, etc. CA "executes" a plan by reminding the user to perform actions. As the state is updated, CA's EXECUTOR revises its reminders. If the user's date is delayed, they may need to rebook dinner. If the user's friend gives them $50, they can skip going to the bank.

As noted in Section 1, TPOPEXEC is able to seamlessly re-execute or omit portions of a plan. This can create ambiguity in the interpretation of standard STNs, as illustrated by the "heating & eating" example. It is also the case that many temporal constraints are more compellingly expressed as constraints between state properties and actions (e.g., "Be at the movie at least 15 minutes before it starts."). These two desiderata serve as motivation for our new language.

3.1 Temporal Constraints

Inspired by PDDL3.0 and linear temporal logic [Gerevini *et al.*, 2009; Pnueli, 1977], our language introduces four temporal modal operators ranging over the actions of our TPOP, the fluents in our planning domain, and time (the positive reals). Our language supports the specification of a set of constraints, but no connectives. During execution, actions in our TPOP may be repeated or skipped, requiring a formalism strictly more expressive than STNs: in an STN, there is no accommodation for unplanned re-execution or omission of actions, nor is there a facility to express temporal constraints with respect to the state of the world [Dechter *et al.*, 1991].

Definition 1. Temporal Constraint Types

- **(latest-before** $b\ a\ l\ u$**)**: A *past constraint* between actions a and b over bounds $l, u \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ stipulates that if b occurs, then a must have occurred previously and the most recent occurrence of a is between l and u time units.
- **(earliest-after** $a\ b\ l\ u$**)**: A *future constraint* between actions a and b over bounds $l, u \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ stipulates that if a occurs, then b must occur in the future and the next occurrence is between l and u time units.
- **(holds-before** $a\ f\ l\ u$**)**: A *past fluent constraint* between an action a and fluent f over bounds $l, u \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ stipulates that if a occurs, then f must have held between l and u time units in the past.
- **(holds-after** $a\ f\ l\ u$**)**: A *future fluent constraint* between an action a and fluent f over bounds $l, u \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ stipulates that if a occurs, then f must hold between l and u time units in the future.

The notation mirrors the PDDL3.0 preference syntax: e.g., **(latest-before** $b\ a\ l\ u$**)** should be read as "the latest occurrence of action a before an occurrence of b is between l and u time units". For the CA example, consider the temporal constraint, **(latest-before** $exercise\ eat\_meal$ 30 240**)**: exercise must be more than a half hour and no more than four hours

after eating. If, after exercising, the user decides to exercise again (due to an exogenous change in the world), the timing of the second exercise action must be consistent with the constraint and the timing of the most recent meal. If the constraint had been **(earliest-after** eat_meal $exercise$ 30 240**)**, there would be no constraint between the meal and the second occurrence of exercise: that constraint is only relevant to the *earliest* occurrence of exercise after a meal.

If $a_{\vdash}$ and $a_{\dashv}$ denote the instantaneous actions corresponding to the start and end of a durative action a, we augment the domain with **(latest-before** $a_{\dashv}$ $a_{\vdash}$ l u**)** where l and u are lower and upper bounds on the duration of a.

3.2 Semantics

We define the semantics of our temporal constraints with respect to the execution trace of the plan – a history of action-state pairs, indexed by time and represented as a *timed word* [Alur and Dill, 1994].

Definition 2. Trace

Given a TPOP $\langle\langle \mathcal{A}, \mathcal{O}\rangle, \mathcal{C}\rangle$ and planning problem $\langle F, O, I, G\rangle$, we define a *trace*, $\mathcal{T}$, to be a finite timed word, $(\sigma_0, t_0), \cdots, (\sigma_n, t_n)$, where $\sigma_i \in \Sigma$, the alphabet Σ ranges over $\mathcal{A} \times \mathcal{S}$, and the time values, $t_i \in \mathbb{R}_{\geq 0}$, are strictly increasing. $\mathcal{T}$ is *executable* iff for every $((a_i, s_i), t_i)$ in $\mathcal{T}$, a_i is executable in s_i. $\mathcal{T}$ is *static* iff for every $((a_i, s_i), t_i)$ in $\mathcal{T}$, if $i > 0$ then s_i is the result of executing a_{i-1} in state s_{i-1}. We signify the concatenation of traces $\mathcal{T}$ and $\mathcal{T}'$ as $\mathcal{T} \cdot \mathcal{T}'$.

We assume that the state of the world is fully observable. If a fluent changes unexpectedly (e.g., through an exogenous event), a tuple in the trace reflects this change. A single tuple $((a_i, s_i), t_i) \in \mathcal{T}$ is an *occurrence* , and we refer to the actual trace of performed actions as an *execution trace*. An execution trace is *valid* if it is executable, satisfies every temporal constraint, and the final action is a_G (i.e., the goal is achieved). Finally, we say that an execution trace is a *valid partial trace* if it can be extended to be a valid trace.

Figure 1 defines the semantics of our temporal modal operators with respect to a trace. We use the abbreviation **(time-diff** i j l u**)** $\stackrel{\text{def}}{=} l \leq t_j - t_i \leq u$ to indicate that the time between indices i and j is bounded between l and u; **(occ** a i**)** to denote that action a occurred at t_i in the trace. For both variants of a future constraint, $\mathcal{T}$ may be a valid partial trace but not a valid trace because the constraint is not yet satisfied – e.g., for **earliest-after**, a appears in $\mathcal{T}$, but b has not occurred since then. Such constraints are *unresolved*.

4 Generalizing and Executing TPOPs

Typical EMs execute actions in a plan in the order prescribed, until the goal is reached or a discrepancy is detected. At that point, they trigger replanning or rescheduling [Lemai and Ingrand, 2003; Conrad and Williams, 2011; Levine, 2012]. The dispatching of STNs operates in a similar fashion [Dechter *et al.*, 1991]. In contrast, TPOPEXEC executes the first action of the cheapest valid plan fragment whose starting state satisfies the necessary conditions for plan validity and satisfies the temporal constraints. Such robustness is not found in existing methods without explicit replanning or rescheduling.

Our approach is to provide a flexible representation that generalizes a plan to capture, for each step of the plan, the necessary subset of state required to ensure the plan's validity. In the case of a TPOP, which compactly encodes k linearizations (sequential plans of length n), this generalization produces up to kn sequential plans of lengths ranging from 1 to n, each leading to the goal but starting in different states. The generalized TPOP is represented as a policy, and it allows for the choice of any one of the (up to) kn sequential plans, filtering out those that do not respect the temporal constraints. Our policy enables TPOPEXEC to choose between the execution of different linearizations depending on the state of the world. In doing so, it can accommodate a number of unanticipated changes, either calamitous or serendipitous.

TPOPEXEC is comprised of an offline preprocessing phase (COMPILER) and an online computation phase (EXECUTOR). COMPILER systematically computes every possible temporally consistent partial plan that corresponds to a suffix of the input TPOP. EXECUTOR retrieves a temporally consistent partial plan that it can use to achieve the goal. To simplify the exposition, we present our approach for a subclass of TPOPs where the constraints, $\mathcal{C}$, are restricted to involve only actions (i.e., the **latest-before** and **earliest-after** constraints): referred to as an ATPOP. In Section 4.3, we show how to express an arbitrary TPOP as an ATPOP. We assume that the ATPOP is provided to TPOPEXEC. Potential sources of an ATPOP include manually hand-coding one, annotating a standard POP with temporal constraints, or computing one with a dedicated planner. For this work, however, we focus on executing an ATPOP rather than its synthesis.

4.1 COMPILER: Offline Generalization

Given an ATPOP, execution trace, and state of the world, TPOPEXEC needs to determine if any fragment of the ATPOP can achieve the goal and satisfy all of the temporal constraints while taking the trace so far into account. We compile the causal and temporal conditions required for every partial plan into a policy that indicates if we can still reach the goal, and if so, what action to execute next and when.

The key component of the policy representation is a *partial plan context*. Given an ATPOP, a partial plan context captures a subset of the original ATPOP actions and orderings, a candidate action, a, and a set of sufficient conditions, ψ, both for the execution of a and to guarantee that some linearization of the ATPOP suffix starting from a will lead to the goal, ignoring for the moment the temporal constraints.

Definition 3. Partial Plan Context

Given a problem $\langle F, O, I, G\rangle$ and ATPOP $\langle\langle \mathcal{A}, \mathcal{O}\rangle, \mathcal{C}\rangle$, we define a *partial plan context* as a tuple $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}, \psi, a\rangle$, where:

1. $\mathcal{A}_{\dashv} \subseteq \mathcal{A}$ is the set of actions to be executed.
2. $\mathcal{O}_{\dashv}$ is a set of ordering constraints over $\mathcal{A}_{\dashv}$.
3. $\psi \subseteq F$ is a set of fluents sufficient for executing $\mathcal{A}_{\dashv}$.
4. $a \in \mathcal{A}_{\dashv}$ and $\nexists a' \in \mathcal{A}_{\dashv}\ s.t.\ (a' \prec a) \in \mathcal{O}_{\dashv}$

$((a_0, s_0), t_0), \cdots, ((a_n, s_n), t_n) \models$ **(latest-before** $b\ a\ l\ u$**)**
iff $\forall j : 1 \leq j \leq n$ if **(occ** $b\ j$**)** then $\exists i : 0 \leq i < j,$ **(occ** $a\ i$**)** $\wedge$ **(time-diff** $i\ j\ l\ u$**)** $\wedge\ \forall k : i < k < j,\ a_k \neq a$

$((a_0, s_0), t_0), \cdots, ((a_n, s_n), t_n) \models$ **(earliest-after** $a\ b\ l\ u$**)**
iff $\forall i : 0 \leq i \leq n-1$ if **(occ** $a\ i$**)** then $\exists j : i < j \leq n,$ **(occ** $b\ j$**)** $\wedge$ **(time-diff** $i\ j\ l\ u$**)** $\wedge\ \forall k : i < k < j,\ a_k \neq b$

$((a_0, s_0), t_0), \cdots, ((a_n, s_n), t_n) \models$ **(holds-before** $a\ f\ l\ u$**)**
iff $\forall j : 1 \leq j \leq n$ if **(occ** $a\ j$**)** then $\exists i : 0 \leq i < j,\ (f \in s_i) \wedge [\exists t^* : (t_i \leq t^* \lneq t_{i+1}) \wedge (l \leq t_j - t^* \leq u)]$

$((a_0, s_0), t_0), \cdots, ((a_n, s_n), t_n) \models$ **(holds-after** $a\ f\ l\ u$**)**
iff $\forall i : 0 \leq i \leq n-1$ if **(occ** $a\ i$**)** then $\exists j : i < j \leq n,\ (f \in s_j) \wedge [\exists t^* : (t_j \leq t^* \lneq t_{j+1}) \wedge (l \leq t^* - t_i \leq u)]$

Figure 1: Semantics of the temporal modal operators with respect to a trace. $l, u \in \mathbb{R}_{\geq 0} \cup \{\infty\}$, $l \leq u$, and a, b are actions.

Context viability captures the notion that there exists a linearization of the partial plan context's POP that is valid and satisfies every constraint. Formally, given a planning problem Π, ATPOP $\langle\langle \mathcal{A}, \mathcal{O}\rangle, \mathcal{C}\rangle$, valid partial trace $\mathcal{T}$, and current state of the world s, a partial plan context $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}, \psi, a\rangle$ is (1) *causally viable* wrt. Π and s iff the POP $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}\rangle$ has a linearization starting with a that is a valid plan for the planning problem with s as the initial state, (2) *temporally viable* wrt. $\mathcal{C}$ and $\mathcal{T}$ iff there exists a trace $\mathcal{T}'$ where the actions in $\mathcal{T}'$ correspond to a linearization of $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}\rangle$ and $\mathcal{T} \cdot \mathcal{T}'$ satisfies every temporal constraint in $\mathcal{C}$, and (3) simply *viable* wrt. Π, s, $\mathcal{C}$, and $\mathcal{T}$ iff $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}\rangle$ has a linearization making the context both causally and temporally viable.

Establishing Causal Viability To generate every causally viable context, we appeal to the approach of Muise et al. (2011) which transforms a POP into a policy. As part of their process, they produce a sequence of condition-action pairs where the condition holds in a state iff some linearization of the POP has a suffix that can reach the goal starting with the action. Space prohibits us from a full exposition, but we modify their algorithm in two ways: (1) rather than simply record the condition ψ and candidate action a, we also record the set of actions and ordering constraints to build a partial plan context, and (2) additional ordering constraints are computed to ensure that, when establishing temporal viability, we reason about the correct linearization. Both modifications are primarily for bookkeeping and the soundness of the subsequent steps. Neither modification has a significant impact on the algorithm's performance. The following proposition follows from the proof of correctness of Muise et al.'s causal viability algorithm (Muise et al. 2011, Theorem 2).

Proposition 1. *Every partial plan context, $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}, \psi, a\rangle$, that we produce is causally viable wrt. Π and s iff $\psi \subseteq s$.*

Establishing Temporal Viability Given the temporal constraints, for each partial plan context, COMPILER determines temporal viability by proving consistency of a carefully constructed *context-specific STN* (CSTN).

An STN consists of a set of events and a set of simple temporal constraints. We use X_a to signify an event for action a, and make the distinction between an action a and an event X_a corresponding to an execution of a. A simple temporal constraint restricts the time between two events to be between a pair of bounds: $[l, u]_{X_{a_1}, X_{a_2}} \stackrel{\text{def}}{=} l \leq t(X_{a_2}) - t(X_{a_1}) \leq u$ where $t(\cdot)$ is a mapping of events to time-points. A CSTN contains events corresponding to the scope of the set of simple temporal constraints relevant to the context. The set of relevant simple temporal constraints, with respect to the ATPOP $\langle\langle \mathcal{A}, \mathcal{O}\rangle, \mathcal{C}\rangle$ and the context $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}, \psi, a\rangle$, consists of:

1. Temporal constraints on the unexecuted actions in $\mathcal{A}_{\dashv}$:
$$\{[\epsilon, \infty]_{X_{a_1}, X_{a_2}} \mid (a_1 \prec a_2) \in \mathcal{O}_{\dashv}\}$$
2. Past temporal constraints ending in $\mathcal{A}_{\dashv}$:
$$\{[l, u]_{X_{a_1}, X_{a_2}} \mid \textbf{(latest-before } a_2\ a_1\ l\ u\textbf{)} \in \mathcal{C}, a_2 \in \mathcal{A}_{\dashv}\}$$
3. Future temporal constraints involving only $\mathcal{A}_{\dashv}$:
$$\{[l, u]_{X_{a_1}, X_{a_2}} \mid \textbf{(earliest-after } a_1\ a_2\ l\ u\textbf{)} \in \mathcal{C}, a_1, a_2 \in \mathcal{A}_{\dashv}\}$$

COMPILER stores only those contexts that have a temporally consistent CSTN [Muscettola *et al.*, 1998]. Such a CSTN may or may not lead to a temporally viable partial plan context depending on the actual timing of *occurrences*. To enable a quick, online re-calculation of temporal viability, COMPILER stores the temporal windows between event X_a and events in the CSTN that correspond to actions *outside* of $\mathcal{A}_{\dashv}$. We ignore future constraints with a single action outside of $\mathcal{A}_{\dashv}$, because without knowing if the first action appears in the execution trace, the CSTN should not include it.

4.2 EXECUTOR: Online Execution

Given the state of the world and execution trace, EXECUTOR follows a four step process: (1) retrieve the set of causally viable partial plan contexts, (2) sort the contexts in ascending distance-to-goal, (3) identify the first context that is temporally viable, and (4) return the leading action and its temporal window. To determine temporal viability, given an execution trace and the stored temporal windows for events that have occurred, EXECUTOR uses the following two-step process:

1. If there are unresolved future constraints in the trace, rebuild the CSTN and recheck its consistency.
2. Simulate the execution of past events in the CSTN.

Resolving Future Temporal Constraints For every unsatisfied future temporal constraint **(earliest-after** $a_1\ a_2\ l\ u$**)**,

we have a set of *occurrences* that are the cause for the constraint remaining unsatisfied: the *occurrences* containing a_1 that have happened *after* the most recent *occurrence* containing a_2. If X_{a_1} does not already exist in the CSTN, then EXECUTOR adds event, X_{a_1}, corresponding to the *latest occurrence of* a_1 and includes the simple temporal constraint $[l, u]_{X_{a_1}, X_{a_2}}$. If there is more than one *occurrence* that serves as a reason for the unsatisfied constraint, EXECUTOR adds another event, X'_{a_1}, to the CSTN corresponding to the *earliest occurrence* containing a_1 (with the constraint $[l, u]_{X'_{a_1}, X_{a_2}}$). The remaining *occurrences* containing a_1 can be ignored as they cannot further constrain the CSTN. EXECUTOR then rechecks for consistency to ensure temporal viability.

Simulating Previous Events Using the standard dispatching algorithm for an STN [Muscettola *et al.*, 1998], EXECUTOR tests if a schedule exists for the actions in $A_{\dashv}$ that adheres to all of the temporal constraints and the execution trace. For every event X_a in the CSTN where $a \notin A_{\dashv}$, we have a corresponding latest *occurrence* $((a, s_i), t_i) \in \mathcal{T}$ (start actions for active future temporal constraints also have an associated *occurrence*). EXECUTOR follows the order found in $\mathcal{T}$ to dispatch each event at the time already established, propagating the start times. If EXECUTOR must dispatch an event at a time outside of its temporal bounds, then the network is inconsistent (cf. Theorem 2 of Muscettola et al. (1998)). If the temporal windows remain non-empty, then the process ends with a temporal window for the event corresponding to the candidate action, a. This provides EXECUTOR both with a certificate that the CSTN is consistent, and indicates what should be done: execute a within its temporal window.

Theorem 1. *Given an ATPOP $\langle\langle \mathcal{A}, \mathcal{O}\rangle, \mathcal{C}\rangle$, valid partial trace $\mathcal{T}$, and partial plan context $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}, \psi, a\rangle$, the context is temporally viable iff the context's CSTN is consistent and can be dispatched following the above two steps.*

Proof sketch. For the context to be temporally viable, $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}\rangle$ must have a linearization that corresponds to some trace $\mathcal{T}'$ such that $\mathcal{T} \cdot \mathcal{T}'$ satisfies every constraint in $\mathcal{C}$. The first set of constraints included in the CSTN ensures that any schedule follows a linearization of $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}\rangle$. The CSTN is consistent and dispatchable iff there is a schedule of the actions in $\mathcal{A}_{\dashv}$ that satisfies every constraint. We thus have a candidate for $\mathcal{T}'$ iff the context is temporally viable. □

Combining Proposition 1 and Theorem 1, we can now ascertain how TPOPEXEC leverages a partial plan context:

Theorem 2. *For a given planning problem Π, ATPOP $\langle\langle \mathcal{A}, \mathcal{O}\rangle, \mathcal{C}\rangle$, execution trace $\mathcal{T}$, state of the world s, and partial plan context $\langle \mathcal{A}_{\dashv}, \mathcal{O}_{\dashv}, \psi, a\rangle$, the partial plan context is viable iff (1) $\psi \subseteq s$, (2) the context's CSTN is consistent, and (3) the context's CSTN can be dispatched.*

4.3 Discussion

We have built computational machinery to enable TPOPEXEC to select a next action and the timing of its execution. Following Theorem 2, as long as a suffix of some linearization of the POP can achieve the goal while satisfying all temporal constraints, TPOPEXEC will eventually achieve the goal. Because determining temporal viability requires some amount of reasoning online, EXECUTOR filters first based on causal viability, and then discards the contexts which are not temporally viable. To choose amongst temporally viable contexts, EXECUTOR prefers the context with the best plan quality based on action cost, breaking ties by the minimum temporal distance between the current time and the goal.

The complexity for computing the causally viable contexts online is at worst linear in the number of contexts, but in practice is much smaller – typically linear in the size of the relevant portion of the current state. The complexity of determining temporal viability is at worst polynomial in the size of the CSTN. However, we have identified many heuristic checks that successfully determine, in the overwhelming majority of situations, whether or not the CSTN is temporally consistent. Naively stored, the number of contexts and temporal networks may pose a problem. However, by leveraging the commonality between the contexts and their temporal networks, we were able to drastically reduce the overall storage compared to [Muise *et al.*, 2011] to store both the state and temporal information.

Optimizations We augmented these methods with a number of critical optimizations. Among the most important are the following: (1) When constructing a CSTN, COMPILER keeps only those events in the scope of any temporal constraint in the CSTN while retaining the transitive ordering from all actions in $\mathcal{A}_{\dashv}$ (not every action in $\mathcal{A}_{\dashv}$ must be a part of a temporal constraint). This reduces the size and complexity of the STN. (2) Rather than always doing a full consistency check for testing temporal viability in the presence of open future temporal constraints, EXECUTOR evaluates a number of necessary or sufficient conditions first. EXECUTOR uses a full consistency check only when more efficient checks fail. Space precludes us from detailing the techniques here, but one example is that if no future temporal constraint tightens a lower or upper bound on an unexecuted action, then the previously compiled temporal windows remain valid: if the CSTN was found to be consistent in the offline phase, then it remains consistent as long as the temporal windows are not tightened.

Reformulation to ATPOP The approach described applies only to ATPOPs. We reformulate TPOPs involving fluent temporal constraints into ATPOPs, enabling the elegant application of our approach to arbitrary TPOPs. The reformulation is sound, but incomplete with respect to the **holds-before** constraint. For completeness, we must expand the temporal reasoning to handle disjunctive constraints (i.e., having a **sometime-before** constraint between actions).

We reformulate our fluent temporal constraints to action constraints. As such, for each fluent participating in a temporal constraint, we introduce an auxiliary action a_f with the precondition of $PRE(a_f) = \{f\}$ and no add or delete effects. These actions are used to record the observation of fluents. We then replace the fluent temporal modal operators with suitable counterparts: **(holds-before** $a\ f\ l\ u$**)** (resp. **(holds-after** $a\ f\ l\ u$**)**) is replaced by **(latest-before** $a\ a_f\ l\ u$**)** (resp. **(earliest-after** $a\ a_f\ l\ u$**)**). This modification permits TPOPEXEC to observe necessary facts at a time required. Finally, we require a unique auxiliary action for each fluent constraint, as sharing the auxiliary actions is unsound.

5 Experimental Evaluation

The core contribution of this work is the ability to execute a plan in a world that can change in unpredicted ways, while reasoning about ongoing causal and temporal viability. We accomplish this while avoiding unnecessary replanning, rescheduling, or plan repair. In building on the work of Muise et al. (2011), TPOPEXEC is able to continue executing a POP in exponentially many more states than traditional approaches that execute actions according to a prescribed ordering. Nevertheless, the work of Muise et al. (2011) cannot reason about temporal constraints. Many execution monitoring systems suffer a similar fate. Standard approaches to schedule dispatching, such as Muscettola et al. (1998), are blind to causal viability and the conditions for the executability of actions. Such approaches will only succeed on problems that do not experience obstructive change.

Our evaluation focuses on TPOPEXEC's robustness and ability to avoid replanning. We also evaluate the general properties of TPOPEXEC's behaviour. The first experiment demonstrates the increased capabilities of our approach over restricted forms of our method that improve on existing execution strategies (i.e., STN dispatching), and the second experiment examines the amount of replanning TPOPEXEC avoids. TPOPEXEC is written in Python, and we conducted the experiments on a Linux desktop with a 3.0GHz processor.

IPC benchmarks lack a combination of causal requirements and complex temporal constraints. As such, we tested our implementation on an expanded version of the CA domain that serves to challenge the causal and temporal reasoning, and is representative of what we might find in the real world. In total, there are 19 actions in the plan (11 durative), 8 past temporal constraints, and 4 future temporal constraints. The ATPOP has 18 ordering constraints which result in a total of 49,140 linearizations. The types of un-modelled dynamics include children becoming hungry, laundry being soiled, etc. In addition to being modelled after real-world temporal requirements, the constraints were designed to pose a challenge for TPOPEXEC. For example, often in the CA domain there is ample opportunity for a context to be causally viable but not temporally viable – the actions in the context can achieve the goal, but not without violating some temporal constraint. Such situations challenge the temporal reasoning aspects of TPOPEXEC to find the most appropriate context.

Rate of Success We simulate our CA agent in a world where fluents change unexpectedly in both positive and negative ways (i.e., adding and deleting fluents from the state). TPOPEXEC fails when EXECUTOR determines the goal is no longer causally and temporally achievable. The level of variability in the world is set using parameter α: $\alpha = 0$ corresponds to no changes whatsoever and $\alpha = 1$ corresponds to significant unpredictable change (at least one fluent changes after every action with 99.998% probability). For 20 different values of α, we ran 1000 trials for each approach. The proportion of successful trials is referred to as the *success rate*.

Ignoring the causal requirements and simply dispatching the ATPOP one action after another mirrors STN dispatching, which we argued above would be unsuccessful in most instances. Nonetheless, we test this approach to verify our intuition and evaluate the level of environmental variability that an STN dispatching algorithm can handle. We also present an *Opportunistic* version of TPOPEXEC that we restricted to execute an action at most once. It can, however, skip actions if positive changes allow. Figure 2 shows the success rate for all three approaches for a given value of α.

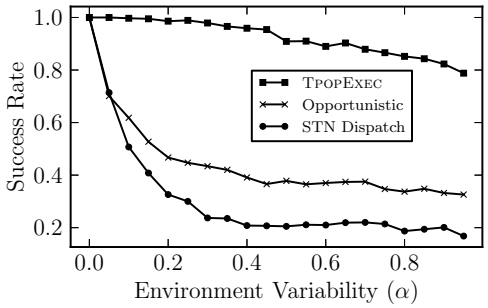

Figure 2: The success rate of the three approaches over a range of environment dynamics, both good and bad.

TPOPEXEC consistently outperforms both the ablated version and STN dispatching by a substantial margin – successfully executing the plan in more than 80% of the instances for almost all values of α and in total succeeding in over 92% of the simulations compared to just over 30% for the STN approach. Having the opportunity to re-execute plan fragments that are required again, while adhering to the imposed temporal constraints, provides us with a distinct advantage. The ablated version, while heavily restricted, still outperformed the STN dispatching for the majority of α-values, solving roughly twice as many instances.

Replan Avoidance We evaluate how often TPOPEXEC avoids replanning during execution in the CA domain. If other systems that replan online are able to replan quickly enough, then their execution behaviour would match ours. Every replan, however, requires solving a PSPACE problem which is what we avoid. Similar to the previous experiment, we evaluate with respect to a range of environment dynamics (the α parameter). However, to properly gauge the need for replanning, we only allow for negative changes to the world: fluents are randomly made false. We count the number of times during execution that TPOPEXEC would be forced to replan if it had not generalized the ATPOP, and we consider only those runs where TPOPEXEC reaches the goal. Figure 3 shows the mean replan rate for a given α-value, normalized by a theoretical maximum number of replans.

We find that the number of replans avoided increases linearly with the increase variability. Due to the temporal constraints on the length of the day, and the length of some durative actions, there is a theoretical limit of roughly 20 replans required for the dynamics we introduce. In situations where TPOPEXEC must operate over a larger time frame, we would expect the potential for replan avoidance to grow.

System Behaviour Profiling EXECUTOR, we found that 35% (resp. 60%) of the time was spent determining if contexts were causally (resp. temporally) viable. The remaining time was used for bookkeeping and data-structure updates for

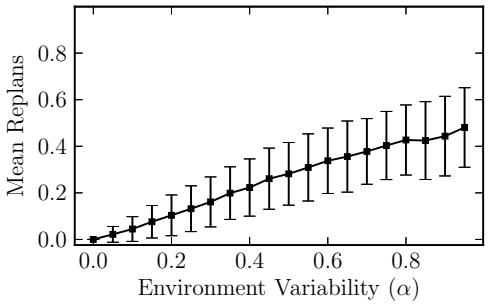

Figure 3: The number of replans that would be required during execution over a range of destructive environment dynamics (mean and standard deviation).

the simulation. COMPILER spent the vast majority of time checking the consistency for the CSTN of each of the 306 partial plan contexts. There is substantial commonality between CSTNs of similar contexts, and a potential optimization is to reuse the computation results. An average of 19 temporal windows were required for every CSTN. However, the memory bottleneck is the data-structure used for computing the causally viable contexts. Using a custom representation, we were able to reduce the memory requirements for our causal information by a factor of four compared to Muise et al. (2011), while our total footprint (causal plus temporal information) used about half the memory of just storing the causal information with the previous representation.

6 Related Work

Most approaches to executing plans with complex temporal constraints assume that an action in a plan will be executed once: an action may appear multiple times in a plan, but each plan appearance corresponds to exactly one action occurrence. IxTeT-eXeC executes actions from a temporally restricted POP and monitors the sufficient conditions for continued causal and temporal viability, replanning when they fail to hold [Lemai and Ingrand, 2003]. In a similar vein, the Pike system executes a temporally restricted POP while continuously monitoring a weaker set of conditions for temporal viability [Levine, 2012]. TPOPEXEC can be seen as an improved executor that tries to avoid plan repair and replanning, and we hope to incorporate our method into a larger system.

The Drake system focuses primarily on executing a plan with complex temporal constraints [Conrad and Williams, 2011]. While it can choose not to execute an action if non-execution is explicitly included as part of a complex temporal constraint, Drake does not represent or reason about causal validity. One avenue we hope to pursue is using the Drake temporal reasoning in place of our CSTNs. Doing so would allow us to handle more expressive constraints.

There is a large body of research on plan execution monitoring (e.g., [Pettersson, 2005; Fritz and McIlraith, 2007; Doherty *et al.*, 2009]). Some systems, such as the work of Doherty et al. 2009, monitor temporal constraints, but many do not. They typically focus on the feasibility of just one partial, sequential plan and resort to replanning when any condition is violated. There have been a number of temporal logics introduced for monitoring plans and schedules (e.g., [Koymans, 1990; Kvarnström *et al.*, 2008]). The most related to our work is TLTL [Bauer *et al.*, 2007]: it uses timed words at the core of its specification and provides a syntax capable of expressing the temporal constraints available to an ATPOP. They do not, however, include a mechanism for deciding what to do next. It is of interest to consider how we might expand our constraint specification language to handle all of TLTL.

7 Summary and Discussion

We presented TPOPEXEC, a system for generalizing and robustly executing a plan that is augmented with temporal constraints. In the face of unexpected changes in the world, TPOPEXEC can select from a large number of valid plan fragments that are consistent with the temporal constraints, repeating parts of a plan or omitting actions, as necessary. This is all done without the need to replan. To accommodate this flexibility, we introduced temporal constraints over actions and fluents, formalizing their semantics with respect to the execution trace. During execution, TPOPEXEC identifies the partial plans, computed offline, that can achieve the goal while satisfying all of the temporal constraints. To choose an action for execution, TPOPEXEC selects one at the start of the best quality partial plan identified as being viable.

We demonstrated our methodology through a prototype implementation and a series of experiments to test the robustness and flexibility of TPOPEXEC. In a simulated uncertain environment for a real-world inspired domain, TPOPEXEC achieved the goal in 92% of the trials while the standard STN dispatching technique succeeded 30% of the time.

We aim to address two fundamental limitations with our work: 1) temporal reasoning and schedule dispatching techniques typically do not consider the state of the world, and 2) execution monitoring schemes for planning problems that allow multiple action occurrences typically do not allow for temporal constraints to be defined. The temporal constraints that we introduce are an essential ingredient for the synthesis of plan execution and schedule dispatching techniques when the environment can change in unexpected ways. They also elucidate the need for referring to both state and actions as integral parts of a temporal constraint.

There may exist a tradeoff between the time saved by avoiding replanning and the quality of a new plan that could be found. In this work, we assume that replanning should be avoided if at all possible, but we hope to consider this tradeoff future work. As the contributions of TPOPEXEC can be seen as complementary to many existing execution monitoring systems (e.g., IxTeT-eXeC or Kirk [Lemai and Ingrand, 2003; Kim *et al.*, 2001]), we hope to incorporate our techniques into a larger system for wider application.

Acknowledgements

We would like to thank the anonymous reviewers whose valuable feedback helped improve the final paper. The authors gratefully acknowledge funding from the Ontario Ministry of Innovation and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[Alur and Dill, 1994] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[Bauer *et al.*, 2007] A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for LTL and TLTL. *Transactions on Software Engineering and Methodology*, pages 1–68, 2007.

[Boutilier and Brafman, 2001] C. Boutilier and R. I. Brafman. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136, 2001.

[Coles *et al.*, 2008] A. Coles, M. Fox, D. Long, and A. Smith. Planning with problems requiring temporal coordination. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pages 892–897, 2008.

[Coles *et al.*, 2010] A. J. Coles, A. I. Coles, M. Fox, and D. Long. Forward-chaining partial-order planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling*, pages 42–49, 2010.

[Conrad and Williams, 2011] P. R. Conrad and B. C. Williams. Drake: An Efficient Executive for Temporal Plans with Choice. *Journal of Artificial Intelligence Research*, 42:607–659, 2011.

[Cushing *et al.*, 2007] W. Cushing, S. Kambhampati, Mausam, and Weld D. S. When is temporal planning really temporal. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 1852–1859, 2007.

[Dechter *et al.*, 1991] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.

[Doherty *et al.*, 2009] P. Doherty, J. Kvarnström, and F. Heintz. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Autonomous Agents and Multi-Agent Systems*, 19(3):332–377, 2009.

[Fox and Long, 2003] M. Fox and D. Long. PDDL2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[Fox *et al.*, 2004] M. Fox, D. Long, and K. Halsey. An investigation into the expressive power of PDDL2.1. In *Proceedings of the 16th European Conference of Artificial Intelligence*, 2004.

[Fritz and McIlraith, 2007] C. Fritz and S. A. McIlraith. Monitoring plan optimality during execution. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling*, pages 144–151, 2007.

[Gerevini *et al.*, 2009] A. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.

[Ghallab *et al.*, 2004] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann, 2004.

[Kim *et al.*, 2001] P. Kim, B. C. Williams, and M. Abramson. Executing reactive, model-based programs through graph-based temporal planning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 487–493, 2001.

[Koymans, 1990] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[Kvarnström *et al.*, 2008] J. Kvarnström, P. Doherty, and F. Heintz. A temporal logic-based planning and execution monitoring system. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, pages 332–377, 2008.

[Lemai and Ingrand, 2003] S. Lemai and F. Ingrand. Interleaving temporal planning and execution: IxTeT-eXeC. In *Proceedings of the ICAPS Workshop on Plan Execution*, 2003.

[Levine, 2012] S. J. Levine. Monitoring the execution of temporal plans for robotic systems. *Master's Thesis*, 2012.

[Muise *et al.*, 2011] C. Muise, S. A. McIlraith, and J. C. Beck. Monitoring the execution of partial-order plans via regression. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 1975–1982, 2011.

[Muise *et al.*, 2012] C. Muise, S. A. McIlraith, and J. C. Beck. Optimally relaxing partial-order plans with MaxSAT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, pages 358–362, 2012.

[Muscettola *et al.*, 1998] N. Muscettola, P. H. Morris, and I. Tsamardinos. Reformulating temporal plans for efficient execution. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, pages 444–452, 1998.

[Pettersson, 2005] O. Pettersson. Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems*, 53(2):73–88, 2005.

[Pnueli, 1977] A. Pnueli. The temporal logic of programs. In *Proceedings of the Eighteenth IEEE Symposium Foundations of Computer Science*, pages 46–57, 1977.

[Smith *et al.*, 2000] D. E. Smith, J. Frank, and A. K. Jónsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1):47–83, 2000.

[Weld, 1994] D. S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27, 1994.

[Younes and Simmons, 2003] H. L. S. Younes and R. G. Simmons. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20:405–430, 2003.